Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology And Methodology

# Predictive analysis in Healthcare

*Supervisor:*
Molnár Bálint
Dr. Habil. Assistant Lecturer

*Author:*
Tshifaro Justin Ndivhuwo
Computer Science MSc

*Budapest, 2023*

Computer Science MSc
1st Semester, 2022-2023 Academic Year

# Thesis Topic Description

Submission deadline: 2022.09.30.

**Student's Data:**

**Name:** Tshifaro Ndivhuwo Justin
**Neptun Code:** SUQG0J

**Course Data:**
**Student's Major:** Computer Science MSc
**Semester & Academic Year of Student's Admission:** 2021/22/1

## A brief description of the thesis topic:
***(Consulting with your potential supervisor, give a synopsis of your planned thesis in a few sentences.)***

Predictive analysis in Health care
The goal of predictive analytics is to determine the likelihood of future outcomes based on historical data. It does this by using data, statistical algorithms, and machine-learning approaches. The objective is to provide the best prediction of what will occur in the future, rather than simply knowing what has already occurred. Predicting the future with the aid of machine learning algorithms is no longer a challenging task, particularly in the field of medicine where it is now possible to predict diseases and anticipate a cure. An overview of the development of big data in the healthcare system will be provided in this paper, and a learning algorithm will be applied to a set of health data. This dissertation aims to predict chronic diseases and mental health using Clustering and Classification models. Both Disadvantages and Advantages Disadvantages will be discussed.

During the experimental design and development, adequate methodologies are used (Design Science Research, Software Case Study, Data Science data and system modeling, etc.). The developed model will be verified and validated by domain-specific metrics.

Thesis Supervisor: Dr. Bálint Molnár

**Specialisation:**
Data Science

Subject related to the topic of the thesis:

Intorduction to Data Science

Advanced Machine Learning

Open-source Technologies for Real-time Data Analytics

# Contents

## Abstract

# Chapter 1

# Introduction

The healthcare industry is one of the most important sectors in the world, with the primary goal of delivering the best possible treatment to patients. With the advancement of data analytics and machine learning, predictive analysis has become an important element of healthcare[1], allowing valuable insights to be extracted from massive amounts of medical data. Predictive analytics techniques rely on historical data to create models that can forecast future results, allowing healthcare providers to make more educated decisions about diagnosis, treatment, and patient care [2].

Because of the broad implementation of electronic health records and other digital health technologies, the volume of healthcare data created has increased rapidly in recent years. The healthcare sector has both opportunities and challenges due to this massive amount of data. On the one hand, data availability presents enormous opportunities for enhancing patient outcomes and healthcare operations. On the contrary, the enormous amount and complexity of healthcare data make extracting valuable insights from it difficult. Predictive analytics is an effective technique for assisting healthcare workers in making sense of this massive volume of healthcare data. Predictive models can find patterns and predict future outcomes by evaluating previous data. Predictive models have many uses in healthcare, including disease outbreak prediction, identifying people at risk of developing particular illnesses and assessing therapy effectiveness[3].

XGBoost is a well-known machine learning method that is widely utilized in healthcare predictive modeling due to its accuracy, speed, and scalability [4]. XGBoost, like any other algorithm, has limitations, and its performance can be enhanced further by optimization techniques. Particle Swarm Optimization (PSO) and the Firefly Algorithm are two swarm intelligence-based optimization algorithms that have shown significant potential in improving the performance of machine learning systems [5]

Swarm intelligence is a metaheuristic optimization technique that solves complicated optimization problems by simulating the collective behavior of social creatures such as ants, bees, and birds. PSO and the Firefly Algorithm, both inspired by the behavior of fireflies and birds, are capable of traversing the search space more efficiently than classic optimization methods [6]. We aim to improve the predicted accuracy of the XGBoost algorithm for healthcare applications by using the power of swarm intelligence.

## 1.1 Motivation and Objectives

The XGBoost algorithm has demonstrated exceptional performance in predictive modeling applications, making it a preferred option for predictive analysis in the healthcare industry. Optimizing the hyperparameters of the XGBoost algorithm can, however, substantially boost the method's performance. Application of optimization techniques like PSO and Firefly to the XGBoost algorithm in the healthcare industry might result in large increases in predicted accuracy. These techniques have shown great promise in improving the performance of machine learning models.

The primary goal of this thesis is to investigate the potential of optimization techniques to improve the XGBoost algorithm's performance for healthcare predictive modeling. We seek to increase the precision and accuracy of the predictive models by applying PSO and Firefly methods to optimize the hyperparameters of the XGBoost algorithm. The research presented in this thesis will add to the expanding body of knowledge on predictive analytics in healthcare and offer insightful information on the potential of optimization methods to improve the efficacy of machine learning algorithms. In the end, this study attempts to assist medical practitioners in making knowledgeable choices regarding patient care, diagnosis, and treatment.

## 1.2 Literature Review

# Chapter 2

# Background and Problem Understanding

# Chapter 3

# Methodology

This chapter aims to describe and fully understand the research methods used. This research's primary objectives are to build a predictive machine learning model, train them on health-related datasets, and optimize its hyper-parameters utilizing swarm intelligence techniques, such as Particle Swarm Optimization (PSO), and Firefly Algorithm. The models and algorithms deployed for this project are thoroughly described below.

## 3.1 Classification models

### 3.1.1 XGBoost algorithm

Extreme Gradient Boosting, often known as XGBoost, is an efficient machine learning technique that is frequently applied in data science and machine learning applications. The Gradient Boosting Decision Tree (GBDT) method, on which it is based, is known for its effectiveness, scalability, and accuracy [7].

The XGBoost algorithm operates by assembling a group of decision trees. It creates decision trees iteratively and combines them to produce a final prediction. XGBoost detects sections of the data where the model performs badly in each iteration and concentrates on improving those areas in the next iteration. This process is repeated until the model reaches the appropriate level of precision.

XGBoost is popular among data scientists and machine learning practitioners due to a number of essential features. Its ability to handle missing data is one of its most crucial qualities. It can also handle both numerical and categorical data and can discover the optimum way to represent categorical variables in the model automatically.

XGBoost is also well-known for its scalability and efficiency. It is designed to take advantage of parallel processing and can handle massive datasets efficiently. XGBoost has also been tuned for performance, including a variety of strategies that reduce memory usage and calculation time.

The adoption of regularization techniques is one of the reasons why XGBoost is so accurate. These strategies prevent overfitting and improve model generalization. XGBoost additionally includes a set of hyper-parameters that can be tuned to improve the model's performance.

To understand the XGBoost well, it is crucial to understand the decision tree first. Figure 3.1 shows the structure of a Decision Tree and figure 3.1 Shows the structure of XGBoost algorithm

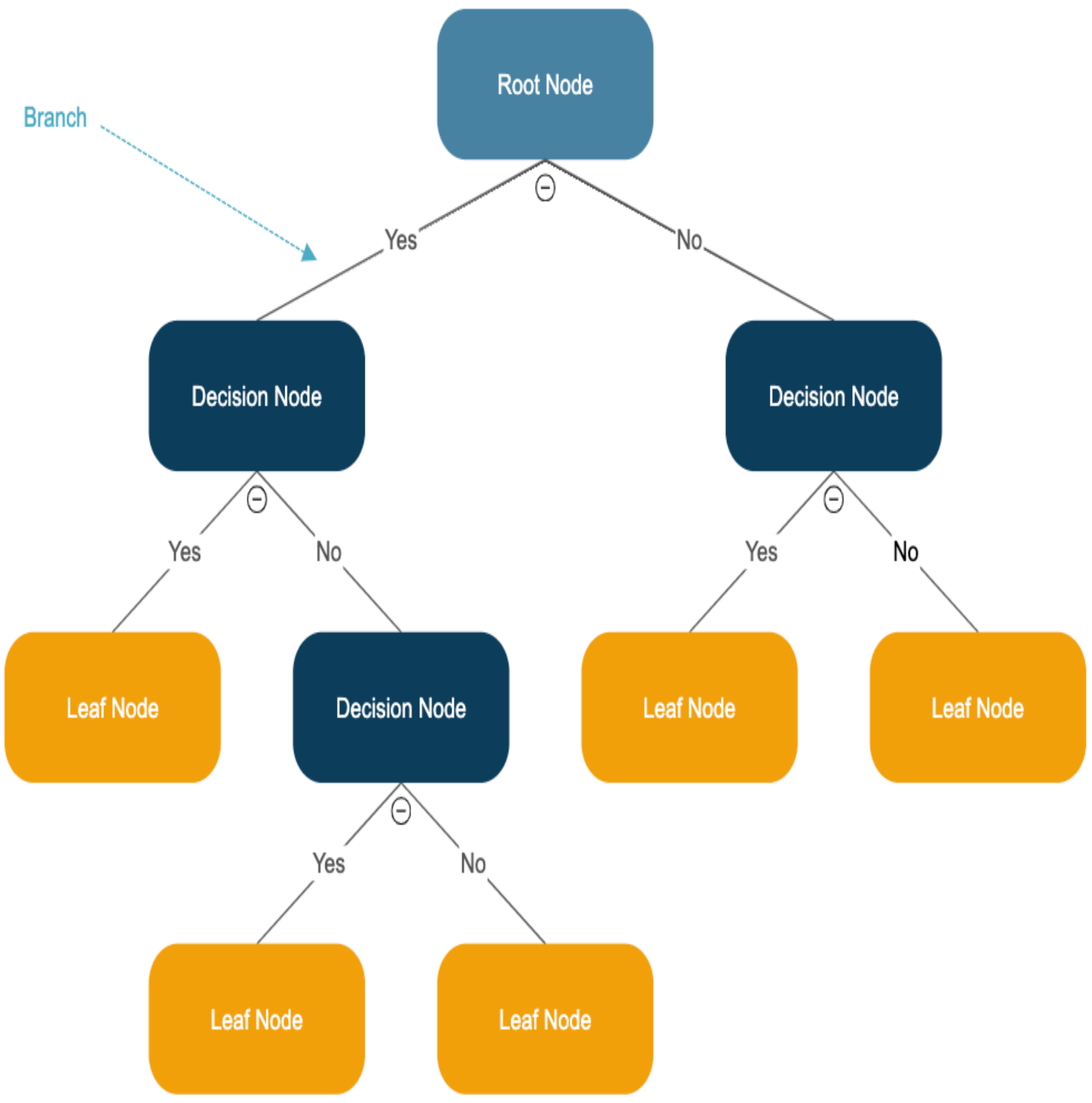

Figure 3.1: Decision Tree Structure
[8]

## XGBoost Hyper-parameters

A hyperparameter is a machine learning parameter whose value is determined before training a learning algorithm.

1. learning_rate This influences the size of the step for each iteration of the boosting procedure. Lower values will make the model more conservative but converge may take longer iterations. Smaller learning rates generally require more trees to be added to the model. It is often referred to as the "shrinkage factor," and it is typically set to a low number, such as 0.1 or 0.01, to guarantee that the model converges slowly and smoothly.

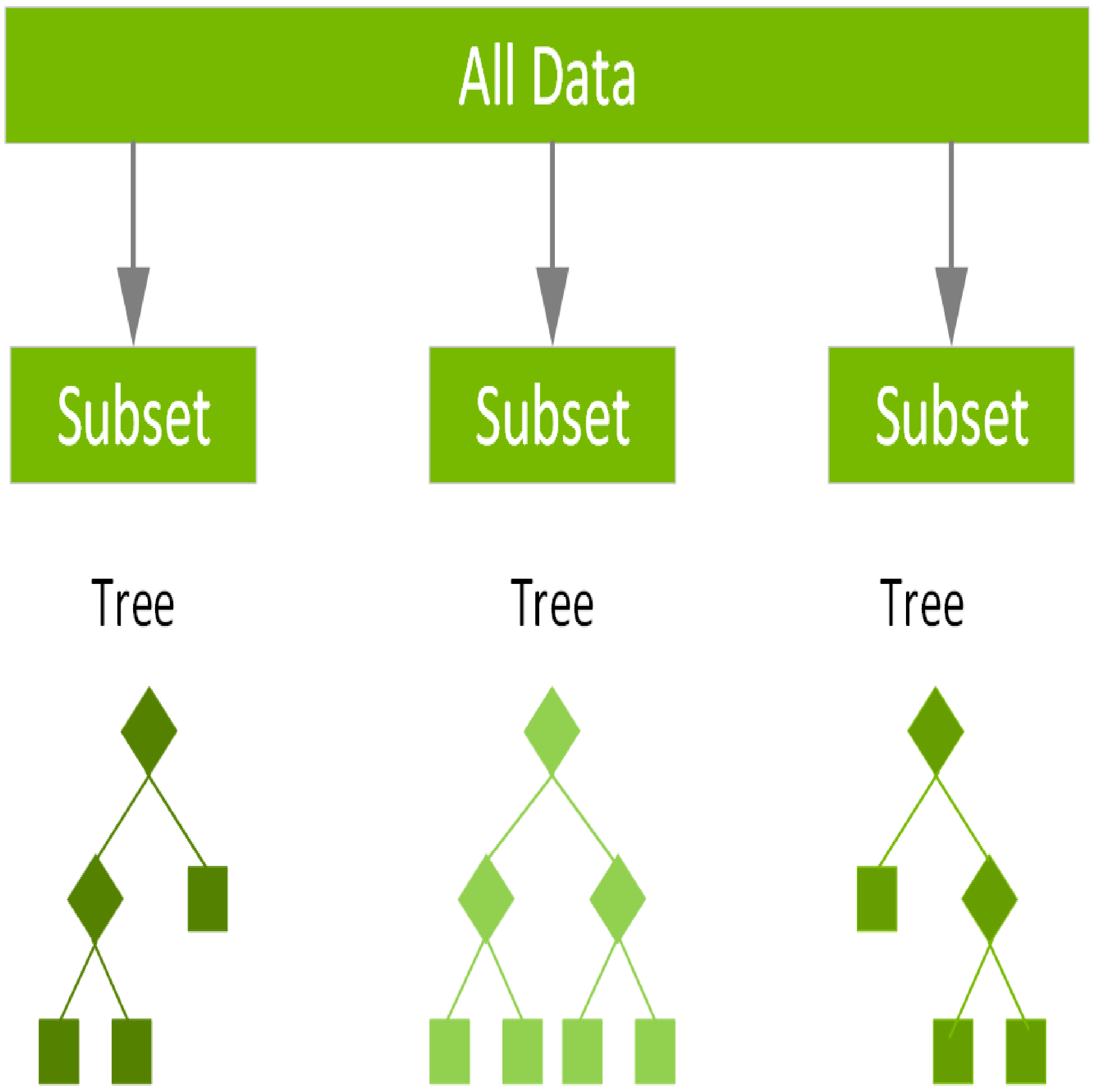

Figure 3.2: Decision Tree Structure
[9]

2. max_depth This regulates each tree in the ensemble's maximum depth. However, deeper trees run the risk of overfitting the data and capturing more complicated interactions. In simple words, it controls the size of decision trees.

3. Subsample This regulates how much of the training data is utilized to create each tree. Smaller numbers will increase the model's resistance to noise, but they can compromise its capacity to detect complex trends in the data. Greater and more complicated models are produced by greater subsample values, which might result in overfitting. This value is typically set between 0.5 and 1.

4. colsample_bytree The colsample_bytree parameter determines how many fea-

tures are used for each tree. A lower colsample_bytree number produces smaller and less complex models, which can aid in the prevention of overfitting. A higher colsample_bytree value leads to larger and more complicated models, which can result in overfitting.

5. n_estimators This determines how many trees are in the ensemble. More trees can improve model accuracy while also increasing the danger of overfitting.

6. min_child_weight This specifies the minimum instance weight required to split a node. Higher values will reduce overfitting and make the model more conservative.

7. gamma This regulates the minimal loss function decrease necessary to split a node. Higher values will reduce overfitting and make the model more conservative.

These are just a few of the many parameters that may be tweaked in XGBoost. There are numerous hyper-parameters, but selecting the optimal mix of parameters for a specific problem takes rigorous experimentation and adjustment, and for these experiments, we chose the hyper-parameters listed above. Table 3.1 shows the recommended and default value of XGBoost Hyper-parameters.It should be noted that these recommended values are not always ideal for every dataset and problem and that a hyperparameter tuning process is recommended to determine the best values for your unique instance.

| Hyper-parameter | Default Value | Recommended Range |
|---|---|---|
| n_estimators | 100 | 100 – 1000 |
| max_depth | 6 | 3 – 10 |
| learning_rate | 0.3 | 0.01 – 0.2 |
| min_child_weight | 1 | 1 – 10 |
| subsample | 1 | 0.5 – 1 |
| colsample_bytree | 1 | 0.5 – 1 |
| gamma | 0 | 0 – 5 |

Table 3.1: XGBoost default Hyper-parameters

### 3.1.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a metaheuristic optimization technique inspired by the social behavior of birds and fish. Kennedy and Eberhart first proposed it in 1995. PSO is a population-based optimization technique that finds the global optimum by simulating the behavior of particles in a multidimensional search space[10].

#### PSO Concept

PSO starts a set of particles at random in the search space, and each particle represents a possible solution to the optimization issue. Each particle moves across the search space by modifying its position based on its own and its neighbors' experiences. Each particle's position is iteratively updated, and the quality of each particle's solution is evaluated using a fitness function[11].

The PSO algorithm determines the direction and speed of each particle's travel using a velocity vector. Each particle's velocity vector is updated depending on two factors: its previous velocity and the difference between its present position and its best position so far. This allows particles to migrate toward better solutions while avoiding becoming stuck in local optima.

Two fundamental parameters define the PSO algorithm: the number of particles and the inertia weight. The size of the swarm is determined by the number of particles, while the inertia weight governs the balance between global and local search. A greater inertia weight favors global search, whereas a lower inertia weight favors local search[12].

#### PSO parameters

The following are the primary parameters used to model the PSO:

1. $S(n) = \{s_1, s_2, \ldots, s_n\}$: a swarm of n particles
2. $s_i$: a swarm individual with position $p_i$ and velocity $v_i, iin[|1, n|]$
3. $p_i$: a particle's location $s_i$
4. $v_i$: a particle's velocity $p_i$
5. $pbest_i$: a particle's best solution

6. *gbest*: the swarm's best answer (Global)

7. $f$ which is the fitness function.

8. $c_1$, $c_2$: Acceleration constants

9. $r_1$, $r_2$: random numbers between 0 and 1

10. $t$: the number of iterations[13]

**PSO Algorithm Details: Mathematical Models**

The PSO algorithm relies on two primary equations. The first equation (3.1) is the velocity equation, in which each particle in the swarm modifies its velocity based on its current position, the computed values of the individual and global best solutions, and other factors. The acceleration factors associated with the personal and social components are denoted by the coefficients $c_1$ and $c_2$. They are referred to as trust parameters because $c_1$ models a particle's confidence in itself, whereas $c_2$ models a particle's confidence in its neighbors [13]. They define the stochastic impact of thinking and social behavior along with the random numbers $r_1$ and $r_2$.

$$v_i^{t+1} = V_i^t + c_1 r_1 (pbest_i^t - p_i^t) + c_2 r_2 (gbest^t - p_i^t) \tag{3.1}$$

The second equation (3.2) is the position equation, where each particle updates its position using the newly calculated velocity:

$$p_i^{t+1} = p_i^t + V_i^{t+1} \tag{3.2}$$

The position and velocity parameters are co-dependent, which means that the velocity depends on the location and vice versa[14].

**PSO execution steps are as follows:**

1. Initialize the algorithm constants.

2. Initialize the solution from the solution space (Initial values for location and velocity).

3. Evaluate the fitness of each particle.

4. Updated individual and global bests (pbest and gbest)

5. Update the velocity and position of each particle.

6. Repeat step 3 until the termination condition is met[13].

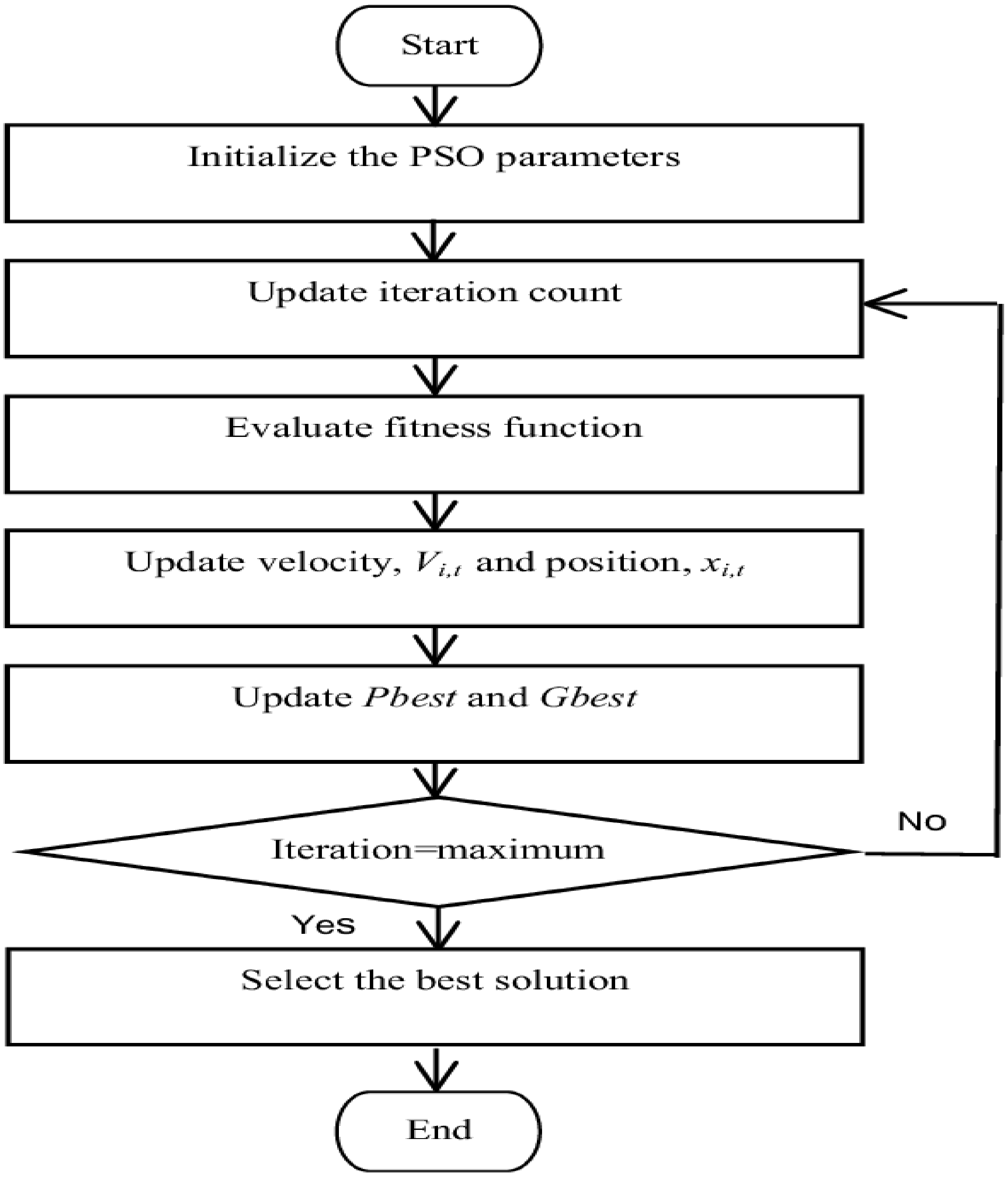
Start
Initialize the PSO parameters
Update iteration count
Evaluate fitness function
Update velocity, $V_{i,t}$ and position, $x_{i,t}$
Update *Pbest* and *Gbest*
Iteration=maximum
No
Yes
Select the best solution
End

Figure 3.3: PSO Algorithm Flowchart
[15]

PSO has been used to solve a variety of optimization problems, including engineering design, scheduling, and data mining. Its benefits include its simplicity, versatility, and capacity to quickly converge on a worldwide solution. However, it is susceptible to premature convergence, and its performance is greatly dependent on parameter selection and problem characteristics[16].

### 3.1.3 Firefly Algorithm

The Firefly algorithm (FA) is a metaheuristic optimization technique inspired by firefly flashing. Xin-She Yang proposed the algorithm for the first time in 2008[17].

The Firefly Algorithm simulates firefly flashing behavior to optimize a given objective function. Each firefly in the algorithm represents a potential solution to the problem being solved. A brightness parameter proportional to the objective function value of the related solution models the flashing behavior of a firefly. A firefly's brightness attracts other fireflies, and the attractiveness reduces as the distance between the fireflies increases[18].

#### Ruless of Firefly Algorithm

Before diving into the rules, keep in mind that all fireflies are unisex, which means that one firefly will be attracted to another firefly regardless of sex and that the brightness of the firefly is determined by the objective function.

The Firefly Algorithm (FA) models fireflies to exhibit the following behavior and rules:

1. Attractiveness: Fireflies are drawn to the light of other fireflies. A firefly's brightness is proportional to the objective function value. Greater brightness implies improved fitness.

2. Movement: Fireflies move at random, however, they prefer brighter fireflies. The stronger the attraction, the brighter the firefly.

3. Intensity The brightness of a firefly is determined by its intensity. The intensity reduces as one moves away

4. Flashing: Fireflies can change the frequency of their flashing to attract others. The strength of the attraction is determined by the frequency of the flashing[19].

### Firefly Algorithm Details: Mathematical Models

The cost function of the given problem determines the firefly's light intensity. The firefly algorithm follows the equation below.

The light intensity reduction abides by the law of inverse square(see (3.3)):

$$I(r) = \frac{I_s}{R^2} \tag{3.3}$$

Where $I(r)$ denotes the intensity at the source and $r$ is the distance from the source

If we consider the absorption coefficient $\gamma$, the light intensity I vary with the square distance $r$.

$$I(r) = I_0 exp^{-\gamma r^2} \tag{3.4}$$

Where $I_0$ is the initial value at (r = 0) and Equation (3.5) expresses the attractiveness, where $\beta_0$ is the beginning value at $(r = 0)$:

$$\beta = \beta_0 e^{-\gamma r^2} \tag{3.5}$$

Equation (3.6) calculates the Cartesian distance between two fireflies, i and j, at their respective coordinates xi and $x_j$. Where D indicates the problem's dimensionality and $x_{ik}$ is the $k^{th}$ element of spatial coordinate $x_j$ of the $i^{th}$ firefly[20].

$$|r_{i,j}| = |r_i - r_j| = \sqrt{\sum_{k=1}^{D} (x_{i,k} - x_{j,k})^2} \tag{3.6}$$

Finally, equation (3.7) determines the movement of a firefly $i^{th}$ when it is attracted to another ( $j^{th}$) attractive (brighter) firefly [20].

$$x_i(t+1) = x_i(t) + \beta_0 e^{-\gamma r^2}(x_j(t) - x_i(t)) + \alpha\epsilon \tag{3.7}$$

where $x_i$(t + 1) is the firefly $i$ position at iteration $t + 1$ displacement. As can be observed, the first element on the right side of Equation (3.7) is the position of firefly $i$ at iteration t, the second is relative to attraction, and the final is randomization (blind flying if no light) where $\alpha$ is the random walk parameter $[0, 1)$ [20].

### Firefly Algorithm Execution steps and Flowchart

**Execution steps:**

1. Initialize Parameters (Population N, $\gamma, \beta, \alpha$)
2. Generate initial population of n flies
3. Compute fitness (intensity) for each firefly member
4. Check if ($t := 1$ to max iteration)
5. Update the position and fitness (light Intensity) of each firefly
6. Return The Best Solution [21]

The FA algorithm iteratively optimizes a given objective function by changing the positions of fireflies in the search space according to these behaviors and rules. The algorithm begins by randomly initializing a swarm of fireflies and then changes each firefly's position based on its attraction to other fireflies in the swarm. The process is repeated until a stopping requirement (e.g., a maximum number of iterations is achieved or a good solution is found) is met. Figure 3.4 (see below) shows the Flow chart of the Firefly Algorithm.

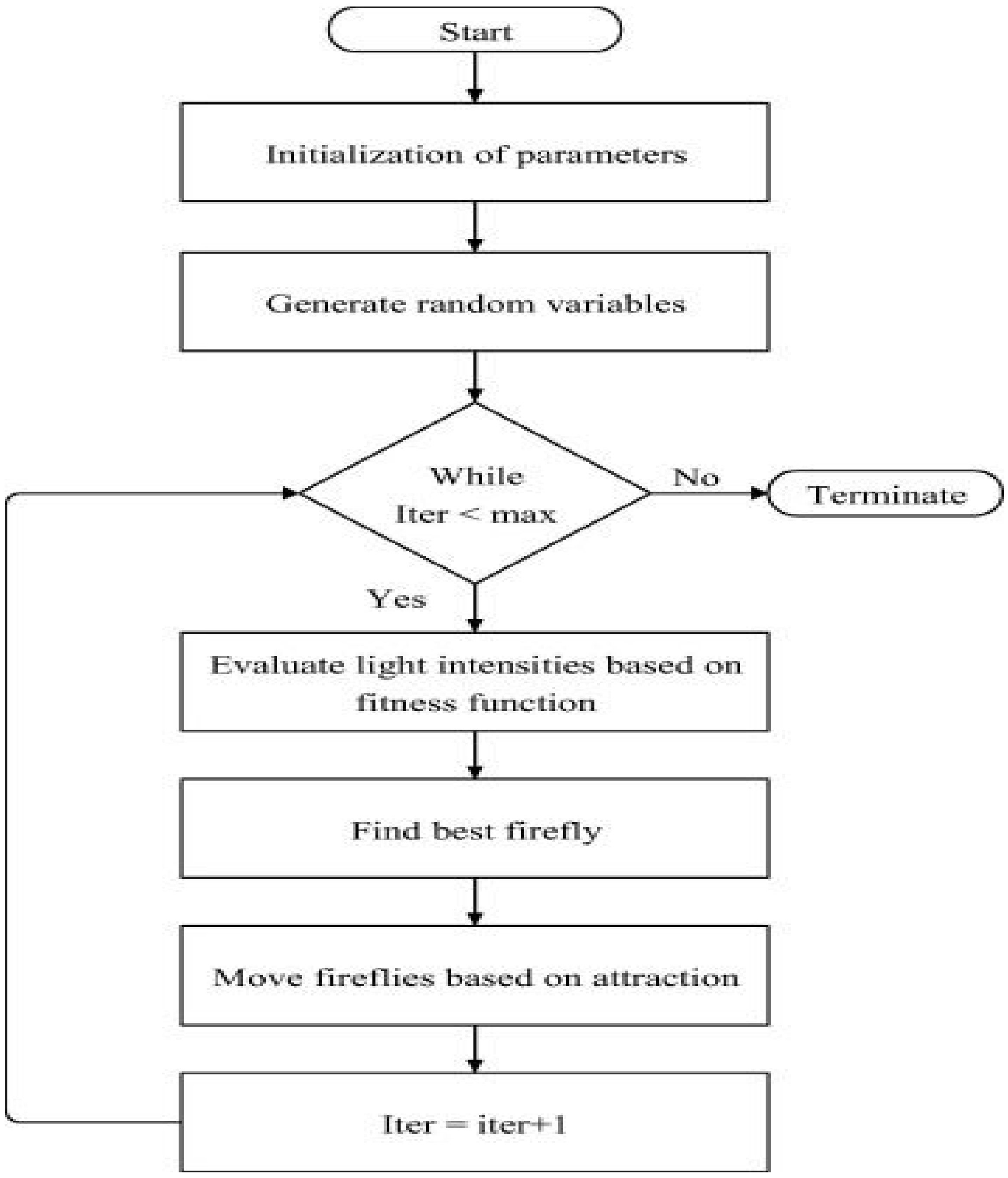
Start
Initialization of parameters
Generate random variables
While
Iter < max
No
Terminate
Yes
Evaluate light intensities based on fitness function
Find best firefly
Move fireflies based on attraction
Iter = iter+1

Figure 3.4: Firefly Algorithm Flowchart
[22]

The Firefly Algorithm has been used to solve a wide variety of optimization issues in industries such as engineering, finance, and healthcare. It has been demonstrated to be effective in discovering high-quality solutions and has piqued the interest of the optimization research community.

# Chapter 4

# Experiments and Results

# Chapter 5

# Conclusion

# Bibliography

[1] Justin Waring, Charlotta Lindvall, and Renato Umeton. “Automated machine learning: Review of the state-of-the-art and opportunities for healthcare”. In: *Artificial Intelligence in Medicine* 104 (2020), p. 101822. DOI: 10.1016/j.artmed.2020.101822.

[2] Frederico H C Ara’ujo, Andr’e M Santana, and Pedro de A. Santos Neto. “Using machine learning to support healthcare professionals in making preauthorisation decisions”. In: *International Journal of Medical Informatics* 94 (2016), pp. 1–7. DOI: 10.1016/j.ijmedinf.2016.06.007.

[3] Pitt SHRS Online. *The Role of Data Analytics in Health Care*. https://online.shrs.pitt.edu/blog/data-analytics-in-health-care/. [Online; accessed 26-April-2023]. 2021.

[4] Yu Liu et al. “A comparison of XGBoost, Random Forest, LSSVM and SVM based on three medical datasets”. In: *International Journal of Online and Biomedical Engineering* 15.11 (2019), pp. 48–66. DOI: 10.3991/ijoe.v15i11.10390.

[5] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016). DOI: 10.1145/2939672.2939785.

[6] Xin-She Yang and Mehmet Karamanoglu. “Swarm Intelligence and Bio-Inspired Computation: An Overview”. In: *Swarm Intelligence and Bio-Inspired Computation*. Elsevier, 2013, pp. 3–23. DOI: 10.1016/B978-0-12-405163-8.00001-6.

[7] T. Chen and Guestrin. “XGBoost: A Scalable Tree Boosting System.” In: *XGBoost* (). URL: https://doi.org/10.1145/2939672.2939785.

[8] Smartdraw. "Decision Tree". In: *J. Object Oriented Program.* 1.1 (Jan. 2023). URL: https://www.smartdraw.com/decision-tree/.

[9] Jason Brownlee. "A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning". In: *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning* (). URL: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/.

[10] Yuhui Shi and Russell Eberhart. "A Modified Particle Swarm Optimizer". In: *Proceedings of the IEEE International Conference on Evolutionary Computation*. 1998, pp. 69–73. DOI: 10.1109/ICEC.1998.699146.

[11] Daniel Bratton and James Kennedy. "Defining a Standard for Particle Swarm Optimization". In: *Proceedings of the IEEE Swarm Intelligence Symposium*. 2007, pp. 120–127. DOI: 10.1109/SIS.2007.368705.

[12] Andries P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[13] Baeldung. *How Does Particle Swarm*. URL: https://www.baeldung.com/cs/pso#3-mathematical-models.

[14] Adrian Tam. *A Gentle Introduction to Particle Swarm Optimization*. URL: https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/.

[15] Zetty N. Zakaria et al. "An Extension of Particle Swarm Optimization (E-PSO) Algorithm for Solving Economic Dispatch Problem". In: *2013 1st International Conference on Artificial Intelligence, Modelling and Simulation* (2013), pp. 157–161.

[16] Russell C. Eberhart and James Kennedy. "A New Optimizer Using Particle Swarm Theory". In: *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 1995, pp. 39–43. DOI: 10.1109/MHS.1995.494215.

[17] Xin-She Yang. "Firefly algorithms for multimodal optimization". In: *In: Stochastic Algorithms: Foundations and Applications, SAGA 2009, Berlin, Heidelberg* (2008), pp. 169–178. DOI: 10.1007/978-3-540-87700-4_14.

[18] Xin-She Yang. "Firefly algorithm, stochastic test functions and design optimisation". In: *International Conference on Modelling, Simulation and Optimization*. 2010, pp. 81–89. DOI: 10.1109/ICMSO.2010.5544696.

[19] Seyedali Mirjalili and Seyed Mohammad Mirjalili. "How effective is the Grey Wolf optimizer in training multi-layer perceptrons". In: *Applied Intelligence* 43.6 (2015), pp. 1503–1517. DOI: 10.1007/s10489-015-0718-1.

[20] Smail Bazi et al. "A Fast Firefly Algorithm for Function Optimization: Application to the Control of BLDC Motor". In: *Sensors* 21.16 (2021). ISSN: 1424-8220. DOI: 10.3390/s21165267. URL: https://www.mdpi.com/1424-8220/21/16/5267.

[21] In: ().

[22] P. Balachennaiah, M. Suryakalavathi, and P. Nagendra. "Firefly algorithm based solution to minimize the real power loss in a power system". In: *Ain Shams Engineering Journal* 9.1 (2018), pp. 89–100. DOI: 10.1016/j.asej.2015.10.005.